Debreceni Egyetem

Informatikai Kar

A celluláris automatákról általában

Témavezető:
Mecsei Zoltán
Számítástechnikai munkatárs

Készítette:
Kovács Andrea Erika
Programtervező informatikus hallgató

Tartalomjegyzék

Előszó

Köszönetnyilvánítás

Meg szeretném köszönni a témavezetőmnek Mecsei Zoltánnak, a hathatós és ötlet teli közreműködést. Illetve ez úton szeretném még egyszer megköszönni azt, hogy a témavezető tanárom lett. Továbbá Laczkó Sándornak és Debreczeni Andrásnak, aki mérhetetlen emberségével segített a mellékeltben található program megírásában. Ezen kívül szeretném még megköszönni mindenkinek, aki érdeklődött a témám iránt, és érdeklődésükkel arra sarkalltak, hogy minél mélyebben ismerjem meg a téma szépségit.

1. Bevezetés

Szakdolgozatom célja, hogy azoknak az érdeklődőknek, akik minimális ismerettel is rendelkeznek az automatákról, nekik egy speciális automatából a sejtautomatából adjak egy kis ízelítőt.

Egy kis történeti áttekintéssel kezdem dolgozatomat. Így szeretném megismertetni az olvasót, hogy mik is ennek az automatának a gyökerei, kik voltak az úttörői. Ha felkeltettem az érdeklődését az automata iránt, akkor tudja, hogy kinek és mely műveiben, cikkjeiben olvashat utána részletesebben.

A következő fejezetet annak szentelem, hogy megismertessem a kedves érdeklődőt a sejtautomaták általános felépítésével, és azokra példát adjak. Így könnyítvén meg a száraz matematikai képletek megértését. A fejezetben azt is látni fogjuk, hogy mikor ekvivalens két sejtautomata.

A negyedik fejezetben találjuk meg egy nagyon híres celluláris automata leírását. Ebben a fejezetben definiálom az automatát, csoportokban rendezem azt a rengeteg mintát, amit az évek során a „játék” rajongói alkottak, és egy két speciális mintát is bemutatok. Ezt követően szétnézünk, hogy mi történt az elmúlt évtizedekben a játékkal. Milyen új szabályok keletkeztek, és milyen új altípusai jöttek létre.

Az ötödik fejezet egy másik nagyon alapvető automatát mutat be. Ezt az egyik úttörő alkotta, és adott ki egy cikksorozatot róla. Itt is bemutatom a matematikai alapokat és utána példát is adok a matematikai képletekre.

A hatodik fejezet a Turing-gépek és a celluláris automaták kapcsolatának megismertetésével telik el. Itt az olvasónak bemutatom a Turing-gépek felépítését, szólok arról egy pár szót, hogy hogyan lehet egy Turing-gépet szimulálni sejtautomatával. Ezen dolgokat először a matematika és az általánosság nyelvén mondom el, majd egy példán keresztül ábrákkal szemléltetve bemutatom ezt a gyakorlatban is.

A következő és egyben utolsó fejezetet egy kicsit komolyabb témának szentelem. Írok a sejtautomaták injektív, szürjektív tulajdonságáról, és ezek a tulajdonságok mire használhatók. Látjuk majd azt, hogy mit jelent az automata megfordíthatósága, milyen egyéb a reverzibilitáson belüli csoportosítása létezik a sejtautomatáknak, illetve hogy a reverzibilitás és a kriptográfia milyen kapcsolatban van egymással.

Ezek után nem maradt más hátra, hogy az olvasónak kellemes időtöltést kívánjak.

2. Történeti áttekintés

Amíg az 1940-es években Stanisław Ulam a Los Alamos National Laboratory-nál dolgozott, tanulmányozta a kristályok növekedését, amihez modellként egy egyszerű rácshálózatot használt. Ugyanekkor John von Neumann, Ulam munkatársa a Los Alamosnál, az önreprodukáló rendszerek problémáján dolgozott. Von Neumann kezdeti tervei egy olyan robotnak az elképzelésén alapultak, ami egy másik robotot épít fel. Ezt a tervet kinematikus modellként ismerik. Ahogy von Neumann fejlesztette a tervét, világossá vált számára, hogy az önreprodukáló robot építésének van egy nagy problémája. A gond az volt, hogy hatalmas költségeket jelent a felépülő másolat tengernyi alkatrésze. Ulam azt javasolta von Neumann-nak, hogy fejlesszen a terve köré matematikai absztrakciókat, mint ahogy azt ő tette ezt a kristálynövekedést tanulmányozásakor. Így született meg a sejtautomaták első rendszere. Mint Ulam rácshálózata von Neumann sejtautomatája is kétdimenziós és az önreplikációt algoritmikusan valósító rendszer volt. Az eredmény egy univerzális másoló és építő lett, ami egy az automatán belüli kis szomszédsággal (csak azok a cellák szomszédok, amik érintkeznek) dolgozó cellánként 29 állapotot megkülönböztető automata lett. Von Neumann bizonyítékot is adott az automata létezésre: egy különleges mintát, ami önmagában végtelen másolatot csinálna az adott sejtes világegyetemen belül. Ezt a tervet a tessellation modellként ismerik, és von Neumann-nak nevezik univerzális konstruktort.

Az 1970-es években széles körben ismertté vált egy két-állapotú, kétdimenziós sejtautomata, ami Game of Life néven terjedt el a korai számítástudományt tanulmányozók közösségben. John Conway találta ki, és Martin Gardner népszerűsítette a [1] Scientific American című folyóirat egyik cikkében. (A játékot lásd később!)

1969-ben emellett a német számítástudomány úttörője Konrad Zuse kiadta a Calulating Space [2] című könyvét, ami azt mondta, hogy a világegyetem fizikai törvényei a természetből fakadóan különállóak, és az egész világegyetem determinisztikus eredménye megadható egy óriási sejtautomata segítségével. Ez a könyv volt az első, ami a ma digitális fizikának nevezett dologról írt.

1983-ban Stephen Wolfram kiadta az első a cikksorozatot [3], ami szisztematikusan vizsgált egy nagyon alapvető, de lényegében ismeretlen típusú sejtautomatát, amit elemi sejtautomatának nevez. Az egyszerű szabályok viselkedésének váratlan bonyolultsága azt a

gyanút keltette Wolframben, hogy ez a bonyolultság a természetben előforduló hasonló szerkezetek miatt lehet. Továbbá ebben az időben, Wolfram megalkotta a belső véletlenszerűség és számítási nem csökkenthetőség fogalmait, és azt állapította meg, hogy a 110-es szabály univerzális lehet — ezt a tény 1990-es években Matthew Cook bebizonyította.

Wolfram az 1980-as évek közepének vége felé elment az akadémiáról ahol addig dolgozott, hogy létrehozza a Mathematica-t. A Matematica egy speciális számítógépes program, amit főleg a tudomány és a matematika használ. Matematikusok és programozók fejlesztették Wolfram-mel közösen. Ez a rendszer az olyan feladatoknak, mint például a szimbolikus- vagy numerikus-számítások, korlátlan precíziós számítások, adatfeldolgozás, vagy görbékkel való ábrázoláshoz nyújt a rendszertámogatást. Mathematica tartalmaz egy programnyelvet is, ami támogatja a funkcionális és procedurális paradigmákat. A Mathematica-t az illinois-i Wolfram Research of Campaign terjeszti. 2002-ben eddig elért eredményeit A New Kind of Science [4] című 1280 oldalas könyvben adta ki, ami alapos érveket hoz fel amellett, hogy a sejtautomatákról tett felfedezések nem elszigetelt tények, de helytállóak, és a tudomány minden ága számára van jelentősége. A sajtóban és az akadémián tapasztalt megdöbbenés ellenére a könyv nem vitatkozik egy a sejtautomatán alapuló alapvető fizikai elmélettel, és bár a könyv leírt néhány sejtautomatán alapuló speciális fizikai modellt, ez szintén minőségileg különböző elvont rendszerekhez modellek bázisát adja.

Rudy Rucker a 2005-ös könyvében, ami a The Lifebox, The Seashell and The Soul [5] címet viseli, kiterjeszti Wolfram elméleteit az Univerzális Automatizmus irányába. Ez modellként arra használta a sejtautomatákat, hogy megmagyarázza azt, hogy az egyszerű szabályok hogyan hozhatnak létre összetett eredményeket.

3. Alapvető definíciók

Vegyünk egy d egész számot. Ez az egész szám fogja meghatározni a teret, amiben az automatát definiáljuk. Ezt a teret jelöljük $\mathbb{Z}^{\mathrm{d}}$-vel. Azt mondjuk, hogy $\mathbb{Z}^{\mathrm{d}}$ elemei lesznek a cellák.

Legyen *S* egy véges állapothalmaz. Az *S* halmaz elemeit állapotoknak fogjuk nevezni. Vagyis egy olyan c celluláris automatát amely *d*-dimenziós és az *S* halmazból veszi fel az állapotait a következő módon írhatom fel:

$$c\colon \mathbb{Z}^{\mathrm{d}} \longrightarrow S.$$

Vezessünk be egy jelölést a cellák állapotaira. A cellák állapotait $\vec{n} \in \mathbb{Z}^d$ jelölhetjük a következő módon:

$$c(\vec{n}).$$

Ezt az összeállítás úgy lehet értelmezni, mint néhány időpillanatban a rendszer összes cellájának állapotáról készült pillanatfelvételét. Elsősorban egy-, illetve kétdimenziós tereket definiálunk, amelyeknél a cellák $\mathbb{Z}$ szerint indexelt sorokat illetve végtelen nagyságú táblát kapunk, amely $\mathbb{Z}^2$ szerint van indexelve. Ezek ismeretében mit mondhatunk az összes konfiguráció számáról? Általánosan véve azt mondhatjuk, hogy a konfigurációk halmaza megadható B^A alakban. Vagyis az általunk definiált c celluláris automatánál az összes konfigurációt leíró halmaz számossága $S^{\mathbb{Z}^d}$. Ez $d = 1$ esetén $S^{\mathbb{Z}}$ konfigurációs halmaz számosságot adja, míg az állapotok halmaza $\mathbb{Z} \longrightarrow S$. Ezek után definiálnunk kell egy szomszédsági vektor is. Azt mondhatjuk, hogy egy d-dimenziós m hosszúságú szomszédsági vektor a következő alakban írható fel:

$$N = (\vec{n}_1, \vec{n}_2, \dots, \vec{n}_m),$$

ahol az igaz, hogy $\forall\, \vec{n}_i \in \mathbb{Z}^d$ és $\forall\, \mathrm{i} \neq \mathrm{j}$ $\vec{n}_i \neq \vec{n}_j$. Az $\vec{n}_i$ elemek minden cella szomszédságának relatív helyét adja meg. Vagyis az $\vec{n}$ cella, amire az igaz, hogy $\vec{n} \in \mathbb{Z}^d$ és m szomszédja van

$$\vec{n} + \vec{n}_i, i = 1, 2, \dots, m$$

alakban írható fel. Ezen kívül ismernünk kell az automata szabályait is. Ezt a matematika nyelvén a következőképpen kell leírnunk: adott számunkra az állapotok halmaza és egy m méretű szomszédság. A szabályokat egy függvény segítségével adhatom meg:

$$f\colon S^m \longrightarrow S,$$

ami azt mondja meg, hogy minden cella új állapotát a szomszédjában lévő cellák régi állapota alapján adhatjuk meg. Ha a szomszédságban lévő cellák állapota rendre $s_1, s_2, \dots, s_m$ volt, akkor a cella új állapota $f(s_1, s_2, \dots, s_m)$ lesz. A celluláris automatákra jellemző, hogy mindegyik cella ugyanazokat a szabályokat alkalmazza, és minden cellára egyszerre teszi ezt. Ez a tény okozza azt, hogy az adott összeállítás globálisan változik. Egy ilyen változást a következő módon lehet leírni: egy c konfiguráció egy c' konfigurációba megy át, ahol az igaz, hogy az $\vec{n} \in \mathbb{Z}^d$

$$c(\vec{n}) = f[c(\vec{n} + \vec{n}_1), c(\vec{n} + \vec{n}_2), \dots, c(\vec{n} + \vec{n}_m)]. \quad (1)$$

Vagyis egy $c \mapsto c'$ leképezés a celluláris automata globális leképezését adja. Ezt a következők módon fogjuk a későbbiekben jelölni:

$$G : S^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d}.$$

A G iterálható, vagyis ismételhető. Ezzel a rendszer időbeli fejlődését idézhetjük elő. Ezt a fejlődést a következő módon adhatom meg:

$$c \mapsto G(c) \mapsto G^2(c) \mapsto G^3(c) \mapsto \cdots$$

Az előbbi felírásban c az evolúció kezdőkonfigurációját jelöli és az

$$orb(c) = c, G(c), G^2(c), G^3(c), \dots$$

sorozat, a c konfiguráció evolúciójának útját írja le.

3.1. Összegezve

Ezeket összegezve, nézzük meg, hogy hogyan is adhatunk meg egy sejtautomatát. Ehhez négy jellemző megadására van szükségünk:

- egy d értékre, ami a dimenziók számát adja meg és igaz rá, hogy $d \in \mathbb{Z}_+$
- egy véges S halmazra, ami az állapotokat fogja tartalmazni
- egy szomszédsági vektorra, ami a következő alakot ölti: $N = (\vec{n}_1, \vec{n}_2, \dots, \vec{n}_m)$
- egy függvényre, ami a szabályokat írja le nekünk: $f: S^m \longrightarrow S$

Ezzel az előbb felsorolt 4 tényezővel megadhatunk formálisan egy celluláris automatát. Vagyis $A = (d, S, N, f)$ négyes egy formális definíció az A celluláris automatára. A globális transzformáló függvény az előbbi négyessel egyértelműen meghatározott, ha figyelembe veszem az (1)-t. Ezt jelölhetem $G(A)$-val vagy csak simán G-vel.

3.2. Mikor ekvivalens két automata?

Egy dolgot kell még megemlíteni. Mikor ekvivalens két automata? Azt mondhatjuk, hogy két automata ekvivalens, ha $G(A) = G(B)$. Teljesen ekvivalens két automata, ha a d dimenziós szám megegyezik és mindkettőnél az állapotokat tartalmazó S halmaz is megegyezik. Viszont a szomszédsági vektor egyezőségét nem megkövetelt.

3.3. Példa a celluláris automatára

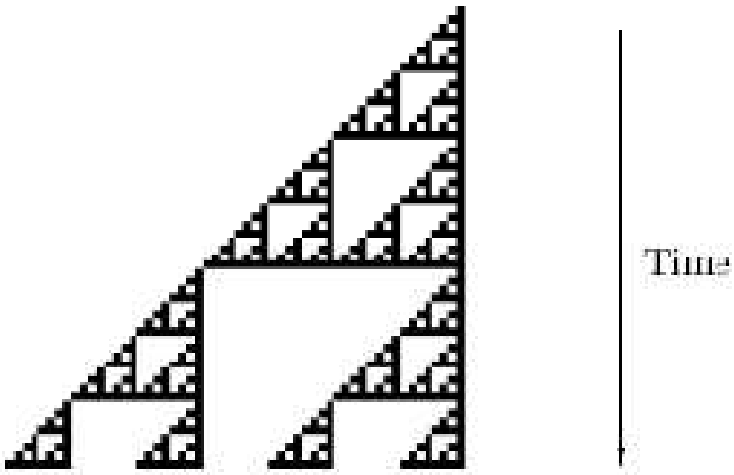
Vegyünk a következő jellemzőkkel rendelkező automatát:

- $d = 1$
- $S = \{0,1\}$
- $N = (0,1)$
- $f: \{0,1\}^2 \longrightarrow \{0,1\}$ ahol $f(a,b) = a + b\ (mod\ 2)$

A cellák egy sorban fognak elhelyezkedni, ezt a $d = 1$ miatt mondhatjuk, ami cella definiálásául szolgáló tér dimenziószáma. Ez a sor a $\mathbb{Z}$ szerint vannak indexelve. Minden cella az alapján változtatja állapotát, hogy összeadja a jobb oldali szomszédja és a saját régi állapotának 2-es maradékát és annak, ha kell, a 2-es maradékát veszi. Ha jobban megnézzük a viselkedését az automatának, akkor XOR logikai műveletre ismerünk rá.

Vegyük egy kicsit jobban szemügyre ezt a példát. Vegyük a következő kezdeti konfigurációt, ami c_0-ként fogunk emlegetni. A kezdő konfigurációban $c_0(0) = 1$ és $\forall\, i \neq 0\ c_0(i) = 0$, vagyis csak a 0. cella lesz 1-es állapotban. Ezután a következő konfiguráció, vagyis a c_1-es a c_0 állítható az előbbiekben megismert G segítségével. Vagyis $c_1 = G(c_0)$ ahol $c_1(0) = c_1(-1) = 1$ és, minden más i-re igaz, hogy $c_1(i) = 0$. Hasonlóan folytatható az időbeli fejlődés, ahol az egyes újabb konfigurációk a következő általános szabály alapján felírhatók:

$$c_j = G\left(c_{j-1}\right)\ ahol\ j = 0, 1, \ldots$$



1. ábra XOR celluláris automata grafikusan

Az 1. ábrán ennek az automatának egy grafikus megvalósítását fogjuk látni. A konfigurációk vízszintesen helyezkednek el. Vagyis egy sorba az adott konfiguráció celláinak állapotait tároljuk. A 0 és 1 értéket a fehér és fekete színű négyzetnek felelnek meg. A legfelső sor a kezdeti $\boldsymbol{c_0}$ konfigurációt tartalmazza, és a rákövetkező sorok rendre az $\boldsymbol{orb}(\boldsymbol{c_0})$ egymás után következő elemeit tartalmazza.

4. Game of Life

Kezdjük a celluláris automaták megismerését egy egyszerű jól ismert példával. John Conway találta ki, és Martin Gardner népszerűsítette. Gardner ezt írta róla:

> „A játék ugyan azonnal híressé tette Conwayt, de feltárta a matematikai kutatás egy egészen új területét, a sejtautomaták területét ... Az élet analógiáján alapulva, az élő szervezetek egy társadalmának az felemelkedésével, hanyatlásával és változtatásaival ez az automata egy olyan növekvő osztályához tartozik, amit szimulánsjátékoknak neveznek. Vagyis olyan játékok, amik igazi életfolyamatokra hasonlítanak."

Az automata egy végtelen négyzetrács hálózaton definiált, vagyis az előző fejezet alapján megállapíthatjuk, hogy $d = 2$. Minden négyzet fekete vagy fehér színt vehet fel, a színek a cella állapotainak felelnek meg. Vagyis az állapotok halmaza $S = \{fehér, fekete\}$. Azt mondjuk, hogy egy cella él, ha fekete színű és halott, ha fehér. Az egész rácshálózat beszínezését az automata egy konfigurációjának nevezzük. Az automata szomszédsági vektora egy 8 hosszú vektor lesz: $N = (\vec{n}_1, \vec{n}_2, \vec{n}_3, \vec{n}_4, \vec{n}_5, \vec{n}_6, \vec{n}_7, \vec{n}_8)$. Az automata egyszerű helyi frissítési szabályokkal rendelkezik, ami magában foglalja azt is, hogy melyik cella fogja az állapotát megváltoztatni. Egy cella új állapotára hatással van a jelenlegi állapota a cellának illetve a körülötte lévő nyolc szomszéd cella státusza is, a következő szabályok szerint:

- Egy fekete, vagyis élő cella, élő marad, ha 2 vagy 3 élő szomszédja van a 8 szomszéd közül.
- Ha 2-nél kevesebb élő társa van az adott cellának, akkor az a cella elszigetelté válik, vagy 3-nál több élő szomszéd esetén túlnépesedetté válik. Ekkor ezek a cellák meghalnak, és fehér szint öltenek.
- Ha egy halott, vagyis fehér cellának, ha 3 fekete szomszédja van, akkor az adott cella feketévé, azaz élővé válik.

Minden cella ugyanazokat a szabályokat használja. A folyamat újra és újra megismételhető és egy időben fejlődő rendszert kapunk általa. A Game of life azért is figyelemre méltó, mert nagyon egyszerű lokálisan frissülő szabályai ellenére a hosszú távú viselkedése kiszámíthatatlan.

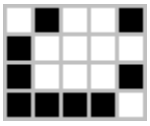
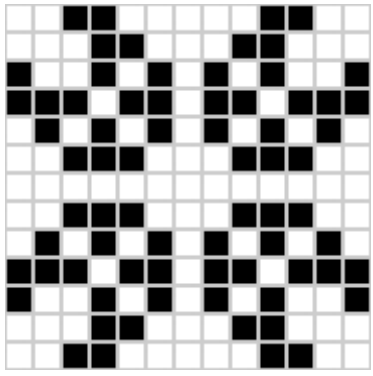
4.1. A minták osztályzása

A Game of Life „rajongói” az évek folyamán a különböző viselkedési mintákat 4 csoportba osztották be:

- csendélet: a szabályok minden cellát ugyanabban az állapotban tartják. A legegyszerűbb ilyen minta az, amikor minden élő cellának két másik élő szomszédja van.
- oszcillátor: ez egy időnként visszatérő minta. Vagyis a szabályok ugyan módosítják a cellák állapotait, de az néhány lépés után visszatér a kezdő mintába ugyanarra a helyre. A csendélet egy speciális típusa.
- űrhajó: a néhány lépés után visszatérő minta, ami mindig a rácshálózat egy eltérő részén jelenik meg. Az oszcillátor egy speciális mozdulatlan űrhajó.
- gun: az oszcillátorhoz hasonlóan véges minta ami periódikusan visszatér a kezdő állapotába, és ezzel együtt „kilő” egy űrhajót magából.

4.2. Példák a különböző osztályokra

Most nézzük meg a minták osztályzása után egy kicsit részletesebben magukat a minták képviselőit.

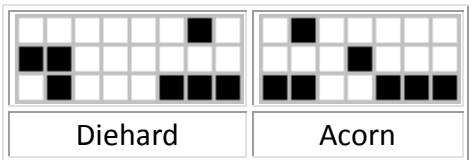
Blokk (csendélet)		Glider (űrhajó)	
Hajó (csendélet)		Könnyűsúlyú űrhajó (LWSS)	
Blinker (két-fázisú oszcillátor)		Pulzár (három-fázisú oszcillátor)	
Toad (két-fázisú oszcillátor)			

1. táblázat Példa a mintaosztályokra

Az 1. táblázatban látott pulzár a legismertebb három-fázisú oszcillátor. Az oszcillátorok túlnyomó többség természetesen két-fázisú, mint a blinker vagy a toad, de ritkán feltűnnek, olyan minták is, amik 4, 8, 15, 30, vagy még több váltás után térnek vissza az eredeti állapotukba.

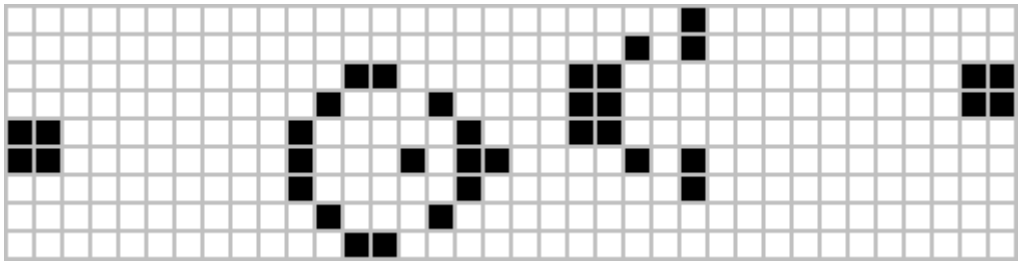
4.3. Néhány különleges minta

Néhány mintát "Methuselahs" hívnak, mert sok lépés telik el, mire a minta visszatér ugyanahhoz az állapothoz, amiből kiindult vagy éppen elhal. A „diehard” vagyis kitartó mintának 130 lépésre van szüksége, ahhoz hogy kihaljon. Az „acorn” nevű mintának pedig mintegy 5200 lépés alatt stabilizálódik egy oszcillátorként és eközben kibocsát még 25 úrhajót is.



2. táblázat Különleges minták

Conway eredetileg azt feltételezte, hogy egy minta sem nőhet korlátlan ideig, azaz bármilyen kezdeti véges számú élő cellával rendelkező konfiguráció működése alatt az élő cellák száma nem halad meg valamilyen véges felső korlátot. Conway azt mondta, hogy annak, aki ezt az előző mondatban olvasott állítást először megdönti vagy alátámasztja, annak 50 dollárt ad, ha azt 1970. december 31-ig megteszi. Az állítás megcáfolására egy mód volt, ha olyan mintát fedeznek fel, ami ismétlődően kibocsát magából egy mozgó mintát, ami nyomot hagy maga után. A díjat novemberben nyerte el a Bill Gospel által vezetett csapat a Massachusetts Technológiai Intézetből. A mintát a csapatvezető után nevezték el Gosper Gun-nak.



2. ábra Gospel Glider Gun

4.4. Variációk a Game of Life-ra

Nézzünk egy pár variációt a Game of Life játékra. A játék eredeti szabályaihoz képest új szabályokat jelentek meg. A hagyományos Game of Life-ban a cella élővé válik, ha pontosan 3 élő szomszédja van, és elő marad, ha 2-3 élő szomszédja van. Egyéb más esetben meghal. Ezt 2,3/3-ként jelzik. Az első szám, vagy számlista, azt jelzi, hogy hány élő cella szükséges ahhoz, hogy élő maradjon, és a második szám vagy számlista pedig, hogy hány élő cella szükséges ahhoz, hogy élő cellává váljon. Az előbbi felíráshoz képest példaként vegyünk most egy másik szabályfelírást: 1,6/6. Vagyis ez azt jelenti, hogy 1 vagy 6 élő szomszéd esetén élő cella marad, és 6 élő cella esetén új cella születik. Egy másik szabályfelírással egy nagyon ismert automatát kaphatunk: 2,3/3,6 ami „HighLife” néven ismert. A HighLife egy továbbfejlesztése a hagyományos Game of Life-nak. Ez a legismertebb replikátor. Több variáció létezik ehhez hasonló automaták definiálására, de ezek nagy része kaotikus, vagy izolált univerzumot definiál. Néhány más variáció módosítja az univerzum geometriáját, és a szabályokat is.

Egy másik változat a bevándorló. Ebben a változatban a Game of Life-hoz hasonló szabályokkal rendelkezik annyi különbséggel, hogy két élőállapot van, ami gyakran nyilvánul meg két különböző színben is. Amikor egy új cella létre jön, akkor azt az állapotot veszi fel, ami a körülötte lévő három cellában megtalálható, ami miatt a cella beszíneződik. Ezt a jellemzőt arra használhatják fel, hogy az űrhajók és más objektumok közötti játékon belüli kölcsönhatást vizsgáljanak. Egy másik hasonló variációt „QuadLife”-nak hívnak, amelynek négy különböző élő állapota van. Ha egy új cella úgy jön létre, hogy három különböző állapot van körülötte, akkor ő a negyedik állapotot veszi fel. Egyéb más esetben a bevándorló új cellájának születési szabályait vallja.

4.5. Kétszemélyes Game of Life

Ismert még a Game of Life kétszemélyes változata is. Ebben a változatban az élő cellának két színe lehet. Az a játékos győz, aki az ellenfél összes élő celláját eltünteti. Egy halott cella, amikor élővé válik, akkor azt a szint veszi fel, ami a szomszédos cellák között domináns, vagyis a bevándorlóhoz hasonlóan pontosan három egyforma színű szomszéd esetén az adott szint. A kezdés véletlen vagy egy előre kiválasztott minta alapján történhet. A játék egy

lépése alatt, a lépésen lévő fél hozzáadhat a táblához egy saját szint, vagy egy ellentéteset elmozgathat onnan.

5. Elemi sejtautomata

Miután megismertük az egyik legismertebb, ha nem a legismertebb celluláris automatát, ezután ismerjük meg egy pár sor erejéig a Wolfram által 1983-ban eléggé kivesézett automata típust az elemi sejtautomatát.

Ez az automata egy egy-dimenziós sejtautomata két állapottal, és egy három hosszú szomszédsági vektorral. Vagyis $d = 1$, $S = \{0, 1\}$, $N = (-1, 0, 1)$ formálisan

$$E = (1, \{0,1\}, (-1,0,1), f)$$

ahol $f: S^3 \longrightarrow S$. A különlegességük az, hogy csak a szabályaikban térnek el egymástól. 256 elemi celluláris automatát ismerünk, mert a különböző helyi szabályok száma $2^8 = 256$. Wolfram bevezetett egy elnevezett sémát, ami azóta szabványossá vált. Azt mondta, hogy minden elemi szabály felírható egy 8 bites sorozatban:

$$f(111)\, f(110)\, f(101)\, f(100)\, f(011)\, f(010)\, f(001)\, f(000),$$

ahol f jelöli az automata szabályait. Ez a bit sorozat egy 0, … ,255-ig terjedő egész számot reprezentál. Ezt a tudomány az automata Wolfram számaként tartja számon.

5.1. XOR

Tegyük fel azt, hogy a 8 bit a decimális 102-t reprezentálja, vagyis 01100110. Tehát az automata Wolfram száma 102, és a szabályok a következők:

$f(111) = 0,$ $f(110) = 0,$ $f(101) = 1,$ $f(100) = 1,$
$f(011) = 1,$ $f(010) = 1,$ $f(001) = 0,$ $f(000) = 0$

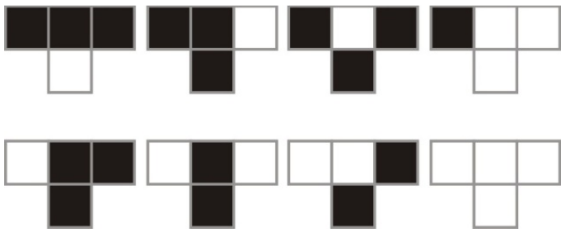
Ez az automata éppen a xor műveletet írja le.

5.2. A 110-es szabály

Tegyük fel, hogy a 8 bit a 110-es értéket tartalmazza binárisan 01101110. Vagyis a van egy elemi sejtautomatánk 110-es Wolfram számmal és a következő szabályokkal:

$f(111) = 0,$ $f(110) = 1,$ $f(101) = 1,$ $f(100) = 0,$
$f(011) = 1,$ $f(010) = 1,$ $f(001) = 1,$ $f(000) = 0$

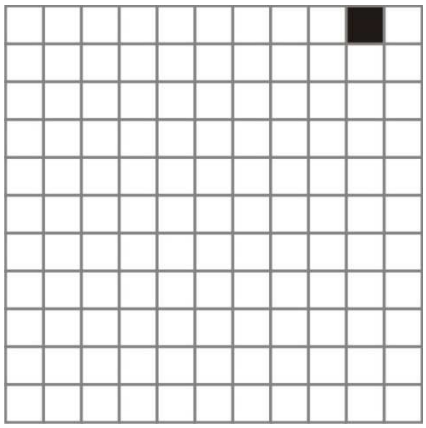
Ássunk egy kicsit a 110-es szabály mélyére. Adottak számunkra az előző két sorban felírt szabályok. Ezeket grafikusan a 3. ábra fogja számunkra megjelenítni. a megjelenítésben minden 1-es bit fekete és 0-s bit fehérrel van jelölve.



3. ábra A 110-es szabály grafikus szabályai

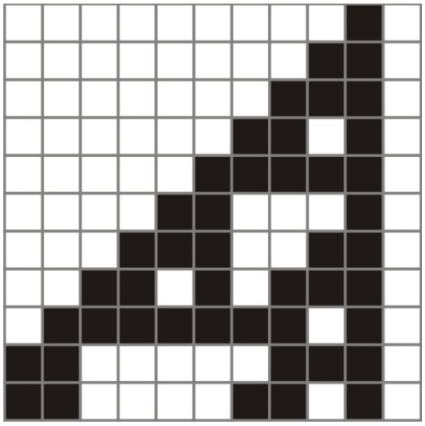
Mit is csinál ez az automata?

Vegyünk a XOR automatához nagyon hasonló konfigurációt. Legyen a tábla, amin az automatát be fogom mutatni 11x11-es. A kezdő konfigurációnkban a tábla első sorában az utolsó előtti elem fekete, azaz 1-es bit értéket hordoz, a többi mind fehér. Ezt a 4. ábra fogja mutatni. alkalmazzuk a megfelelő szabályokat sorról sorra. Most az első soron kell végigmenni úgy, hogy hármasával nézzük a cellákat, és megnézzük, hogy az adott hármas, amit megfogtunk a grafikusan felrajzolt szabályok melyikére illik. Hogy ne legyen ennyire ködös, amit írok, nézzük meg részletesen az első sort



4. ábra A 110-es szabályhoz tartozó automata kezdő konfigurációja

Vegyük az 1. 2. 3. cellákat az első sorban. Látjuk, hogy ezek mind fehérek, ezért meg kell keresni azt a szabályt, ami megmondja, hogy 3 fehér esetén mi teendő. A szabály azt mondja, hogy ez eredmény cella üres lesz, vagyis marad fehér. Ezután vesszük a 2. 3. 4. cellákat az első sorban és megnézzük, mely szabály vonatkozik rájuk. Ugyanaz, mint az előzőre.



5. ábra Az automata szimulálása

Az első eltérés a 8. 9. 10. cellahármas vizsgálatánál lesz, mert ott a második sorban az 9. cella feketére fog színeződni. Hasonló fog történni az 9. 10. 11. cellahármas vizsgálatánál is. Az 5. ábra fogja a teljes kitöltött táblát mutatni.

6. A celluláris automaták és a Turing-gépek

6.1. Pár szó a Turing-gépekről

Némi túlzással azt is mondhatnám, hogy a Turing-gépeket játék számítógépeknek is tekinthetjük. Azért mondhatjuk ezt, mert nagyon egyszerű definiálni őket, és lépésről lépésre hajtják végre a műveleteiket. Egyet viszont tudunk, elég erős ahhoz, hogy korlátlan algoritmusokat szimulálhassunk velük.

Egy determinisztikus Turing-gépet meghatároznak a következő elemek:

- egy véges állapot halmaz, amit jelöljünk Q-val
- egy kezdő egy elfogadó és egy elutasító állapot $q_0, q_a, q_r \in Q$ amelyekre az igaz, hogy $q_a \neq q_r$
- Γ véges szalag ábécé
- $\Sigma \in \Gamma$ bemenő ábécé
- egy olyan b szalag szimbólum, amire igaz, hogy $b \notin \Gamma \backslash \Sigma$
- és egy függvény: $\delta: Q \times \Gamma \longrightarrow \mathrm{Q} \times \Gamma \times \{L, R, H\}$. $\forall \gamma \in \Gamma$ teljesülni kell annak, hogy $\delta(q_a, \gamma) = \delta(q_a, \gamma, H)$ és $\delta(q_r, \gamma) = \delta(q_r, \gamma, H)$.

A géphez hozzá tartozik a szalag és egy fej is. A szalag kétirányú, végtelen hosszan olyan cellákat tartalmaz, amelyekben a szalag ábécé egy-egy szimbóluma található. A szalag cellái az egész számok alapján vannak indexelve. A szalag tartalma minden időpillanatban leírható egy t függvénnyel. A függvényre igaz, hogy a $t \in \Gamma^{\mathbb{Z}}$ és $\forall i\ t(i) \in \Gamma$ vagyis az i-edik cellában lévő szimbólumot adja vissza. A fej egy véges állapotú automata, ami mozgatja és olvassa a szalagot. Adott még ezen kívül rengeteg elem hármas. Ezek $(q, i, t) \in Q \times \mathbb{Z} \times \Gamma^{\mathbb{Z}}$ alakban adhatók meg, és ők tartalmazzák az összes információt a gép jelenlegi állapotáról, ami a q állapot, az i-edik hely a szalagon és a szalagnak megfelelő t. Azt a (q, i, t) konfigurációt, ahol a $q = q_a$elfogadó vagy $q = q_r$ elutasító konfigurációnak nevezzük.
Egy Turing-gép egy lépése alatt - ami függ a fej állapotától és a szalagon található szimbólum olvashatóságától – megváltoztatja az állapot, lecseréli a szalagon lévő szimbólumot egy másikra és jobbra vagy balra mozgatja vagy helyben hagyja a szalagot attól függően, hogy ezt a δ hogyan határozta meg. Ennél egy kicsit pontosabban leírva: a (q, i, t) konfiguráció a

$(q', i + d, t')$ konfigurációba megy át, ha $\delta(q, t(i)) = (q', \gamma, d)$ és $t'(i) = \gamma$ valamint $t'(j) = t(j)$ minden $i \neq j$. A következőképpen is jelölhetjük a fent leírtakat:

$$(q, i, t) \vdash (q', i + d, t').$$

A $\vdash$ jel tranzitív, reflexív lezártjára a következő jelölést is használhatjuk

$$(q, i, t) \vdash^* (q', i', t'),$$

ekkor (q', i', t') levezethető (q, i, t)-ból 0 vagy több lépésen keresztül.
A Turing-gépet a Σ bemeneti ábécé felett értelmezett nyelv felismerésére használjuk a következő módon: minden $w \in \Sigma^*$ szó szalagon való reprezentációja $t_w \in \Gamma^{\mathbb{Z}}$, ahol a w szó szimbólumait sorban a szalagra írtuk $1, 2, \ldots, |w|$ cellákba, és minden más cella csak $blank$ szimbólumot tartalmaz. A gép pontosan akkor fogadja el a w szót, ha

$$(q, 1, t_w) \vdash^* (q_a, i, t)$$

minden $i \in \mathbb{Z}$ és $t \in \Gamma^{\mathbb{Z}}$ esetén, ekkor a gép megáll a q_a elfogadó állapotban. Egyéb esetben a szót nem fogadja el. Itt jegyzem meg, hogy az elutasítás két módon történhet: vagy megáll az elutasító állapotban vagy soha nem áll meg.

6.2. Hogyan „utánozzunk” egy Turing-gépet celluláris automatával?

Miután egy kicsit fogalmunk lett arról mi is az a Turing-gép, nézzük meg, hogyan lehet összefüggésbe hozni a sejtautomatákkal.

Fejezetem célja hogy megvizsgáljuk a sejtautomaták számítási univerzalitását. A nyelvek felismerése a megállási szimbólum, a kezdő, és az elfogadó állapot miatt egyértelmű, vagyis a Turing-gép számítási univerzalitása pontosan meghatározható. A celluláris automaták esetében koránt sem ilyen egyértelmű a helyzet. Nem létezik olyan definíció, ami széles körben elfogadott. A bemenet lekódolását és az elfogadó állapotokat az irodalmak mind különböző módon említik. Célunk érdekében, hogy a celluláris automaták hatékonyságát megmutassuk, nincs is szükségünk arra, hogy a celluláris automata esetén beszéljünk számítási értelemben vett univerzalitásról, egyszerűen a Turing-gép esetén vett megállási

feltételeket használjuk, és egy tetszőleges Turing-gép sejtautomatává való „bijektív” leképezését mutatjuk meg.

Nézzünk egy nyilvánvaló módját a Turing-gép szimulálásának. A következő konfiguráció az M Turing-gépet megvalósító egy dimenziós celluláris automata. A celluláris automata konfigurációja két sávos: az egyik sáv tárolja a szalag tartalmát, amíg a másik sávon az éppen használatban lévő cella tartalma tárolja a Turing-gép állapotát. A Turing-gép minden lépését a celluláris automata szabályai valósítják meg, tehát a változások csak két cellában valósulnak meg: a cellák a Turing-gép lépés előtti és utáni állapotát tartalmazzák.

Legyen az M Turing-gép állapothalmaza Q és Γ a szalag-ábécéje, és δ az átalakító függvény. A sejtautomata S állapot halmaza a következő párosokat tartalmazza:

$$(q, a) \in (Q \cup \{0\} \times \Gamma.$$

Ezekre a párosokra jellemző, hogy $0 \notin Q$ és azokat a párokat tartalmazza, amik az adott cella tartalmán kívül a gép előállíthat. A párok első komponensére igaz, hogy $q \neq 0$ és a q állapotú cella olvasható a gép fejében.

A sejtautomata 1 sugarú szomszédságot használ és a szabályok a következőképpen vannak definiálva:

- ha a szomszédságban nem található meg a Turing-gép feje, akkor a cella állapota nem változik.
- ha a szomszédság egynél többször tartalmazza a gép fejét, akkor sincs változás. Ebben az esetben soha nincs ellenőrzött szimulációja a Turing-gépnek.
- feltételezzük, hogy pontosan egy szomszédsági cellában van jelen a Turing-gép feje. Ekkor csak a következő esetekben megy változás végbe:
    - A cella a (q, a) állapotban van. Legyen $\delta(q, a) = (q', a', d)$. Az új állapota a cellának $(0, a')$.
    - A jobb szomszédja van a (q, a) állapotban és a $\delta(q, a) = (q', a', L)$. Ekkor a cella új állapota (q', x) ahol $(0, x)$ a régi állapot.
    - A bal szomszédja van a (q, a) állapotban és a $\delta(q, a) = (q', a', R)$. Ekkot a cella új állapota (q', x) ahol $(0, x)$ a régi állapot.

Az automata változatlan állapota $(0, B)$ pár ahol a B a Turing-gép megállási szimbóluma.

Most nézzük meg, hogy hogyan lehet a Turing-gép konfigurációját egy sejtautomata konfigurációjává tenni.

Tudjuk, hogy a Turing-gép konfigurációja (q, i, t) hármas ahol minden elemre igaz, hogy $q \in Q, i \in \mathbb{Z}$, és $t \in \Gamma^{\mathbb{Z}}$. Ezt a következő módon fogjuk sejtautomata konfigurációvá tenni: a $c \in S^{\mathbb{Z}}$ az automata egy konfigurációja és minden $j \in \mathbb{Z}$ igaz hogy

$$c(j) = \begin{cases} \big(q, t(j)\big) & ha\ j = i \\ \big(0, t(j)\big) & ha\ j \neq i \end{cases}$$

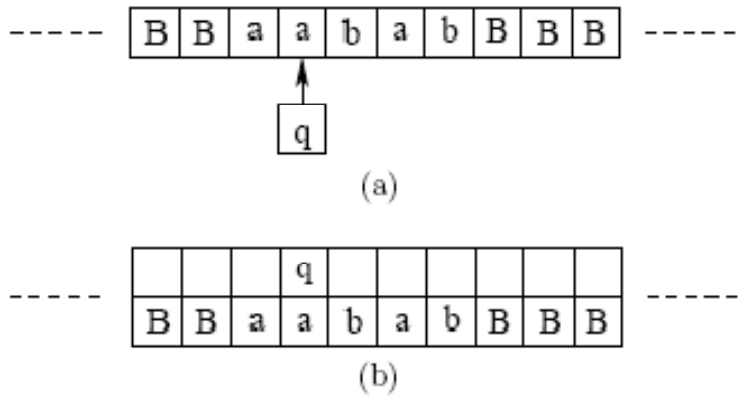
Vagyis miután megtudtuk, hogy hogyan lehet egy Turing-gép konfigurációból sejtautomata konfigurációt csinálni, jó lenne, ha valamilyen függvény ezt elvégezné helyettünk. Erre definiáljuk a következő függvényt, ami előállít egy sejtautomata konfigurációt egy Turing – gép konfigurációból:

$$c = E(q, i, t),$$

ahol az

$$E: Q \times \mathbb{Z} \times \Gamma^{\mathbb{Z}} \longrightarrow S^{\mathbb{Z}}$$

Erre a konfiguráció átkódolásra példa a 6. ábra.



6. ábra Turing-gép (a) konvertálása sejtautomatává (b)

Ezek szerint, ha egy Turing-gép (q, i, t) állapota k lépés alatt változik a (q', i', t') állapotúvá, akkor a celluláris automata is k evolúciós lépés alatt változtatja az $E(q, i, t)$ konfigurációt $E(q', i', t')$ konfigurációvá. Ez a tény azt jelenti, hogy a sejtautomata szimulálja a Turing-gép lépéseit. Vagyis ha az M Turing-gép rendelkezik egy számítási univerzummal, akkor a leképezés eredményeként előálló sejtautomata is rendelkezni fog ugyanazzal az univerzummal. Vagyis a Turing-gép bemeneti szava a sejtautomata kezdő konfigurációja lesz, és a szó elfogadott, ha az automata van olyan cellája (q_a, x) állapotú ahol teljesül, hogy $x \in \Gamma$ és q_a a Turing-gép elfogadó állapota.

6.3. Egy konkrét példa leírása

Most nézzünk egy gyakorlati példát arra, hogy hogyan lehet az előbb látottakat hasznosítani. Ezzel egy példát adok a Turing-gépekre és arra, hogy azt hogyan lehet celluláris automatává alakítani.

Készítsünk egy olyan egyszalagos Turing-gépet, ami a bemenő bináris szót negálja. Ehhez meg kell adni az állapotok halmazát, a szalagábécét, a bemenő ábécét, a kezdőállapotot, a végállapotokat, és a leképezési függvényt. Jelen esetben a szalagábécé $\Gamma = \{0,1,\#\}$, az állapotok halmaza $Q = \{q_0, q_1, q_a\}$, a kezdőállapot q_0, a végállapotok halmaza $\{q_a\}$, a leképzési függvény a következőképpen néz ki:

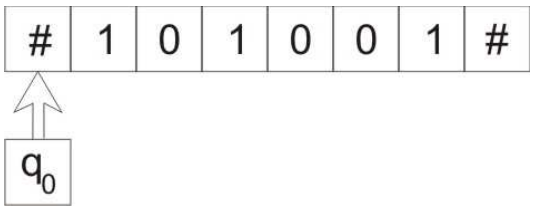
$$\delta(q_0, \#) = (q_1, \#, L) \qquad \delta(q_1, 1) = (q_1, 0, L)$$

$$\delta(q_1, 0) = (q_1, 1, L) \qquad \delta(q_1, \#) = (q_a, \#, H)$$

A szabályok működése a következő:

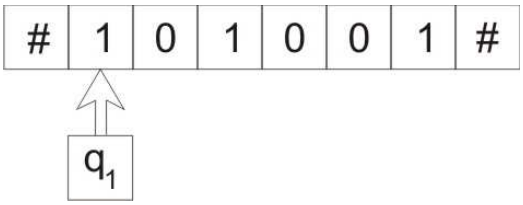
1. Menjen általános állapotba, majd álljon a szó első betűjére.
2. Ha az olvasott betű 1-es, akkor írja át 0ra majd lépjen a következő betűre.
3. Ha az olvasott betű 0, akkor írja vissza, majd lépjen a következőre.
4. Ha az olvasott betű #, akkor írja azt vissza és lépjen végállapotba, különben a 2. pontra.

Legyen most az automatánk kezdő szava #101001#



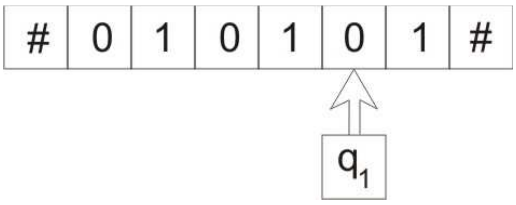
7. ábra A Turing-gép kezdőállapota

Mint látjuk, a gép a q_0 állapotban van, és a # jelet olvassa. Ekkor a $\delta(q_0, \#) = (q_1, \#, L)$ leképzési szabály használható és a 8. ábrán látható állapotot veszi majd fel.



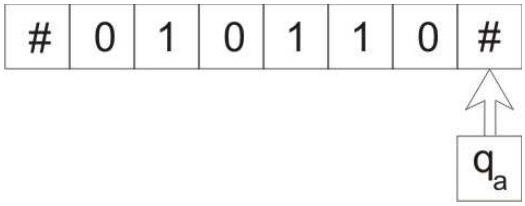
8. ábra A Turing-gép az első lépés után

Ezután sorban a $\delta(q_1, 1) = (q_1, 0, L)$ szabály alkalmazása következik, amely hatására a szalagon a fejnél lévő cellában álló 1-es 0-ra változik, és a szalag balra mozog.



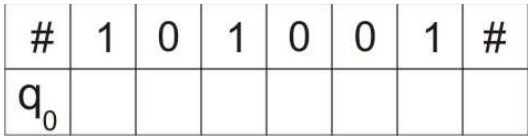
9. ábra A Turing-gép öt lépés után

Az 9. ábra a Turing-gép állapotát fogja mutatni 5 lépés után. A 10. ábra, pedig a Turing-gép megállásakor előálló állapotot fogja mutatni. Ezek után, hogy a gyakorlatban is láttuk, hogy működik egy Turing-gép, nézzük meg, hogy lehet belőle celluláris automatát csinálni. Először is fel kell egy kicsit frissíteni az emlékeinket. Azt mondtuk, hogy a Turing-gép állapota egy hármas, ami egy állapot, a bemeneti ábécé egy eleme és az irány amerre mozog a szalag. Azt mondtuk, hogy ezt át kell valahogy alakítani úgy, hogy a megfelelő cellában a megfelelő szimbólum legyen.



10. ábra A Turing-gép az utolsó lépés után

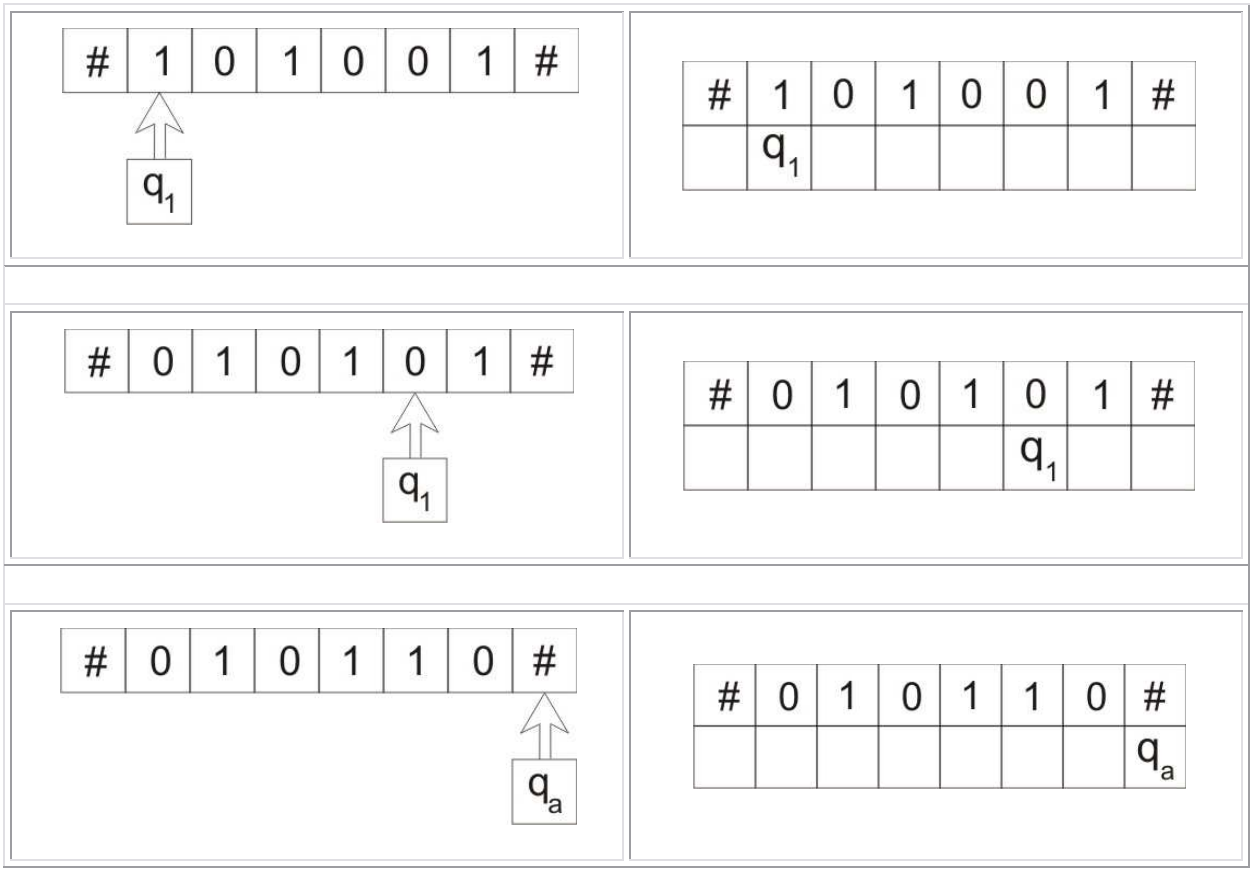
Vagyis a 6. ábrához hasonló szemléltetésünk lesz. Lesz két sávunk. Az egyik sávban tartjuk számon a szalag tartalmát, a másik pedig sáv pedig a Turing-gép állapotának számontartására szolgál. A gép kezdő állapota a $(q_0, \#, L)$ hármas és az automata kezdő konfigurációja $(q_0, \#)$. Az automata szabályai hatásukban megegyeznek a gép szabályaival. A példaként definiált Turing-gépből készített celluláris automata kezdeti állapotát a 11. ábra mutatja nekünk.



11. ábra A Turing-gépet szimuláló celluláris automata kezdeti konfigurációja

Mivel azt állapítottuk meg, hogy ha egy Turing-gép adott állapotból k lépés alatt levezethető másik állapotba jut arra az őt szimuláló celluláris automata is képes. Vagyis az egyes

lépésekben a celluláris automata nagyon hasonló lesz ahhoz a Turing-géphez, amit szimulál. A következő táblázat a 8. 9. 10. ábrát fogja celluláris automata szimulációval:



3. táblázat Turing-gép celluláris automatával szimulálva

Miután láttuk, hogy a celluláris automata tud Turing-gépet szimulálni azt is meg kell jegyezni, hogy ez fordítva is működik.

7. A celluláris automaták megfordíthatósága

Ez a fejezet egy kis betekintést fog adni az automaták megfordíthatóságába. Kezdjük egy kis kiegészítéssel. Definiálnunk kell néhány tulajdonságot ahhoz, hogy meg tudjuk mondani mikor megfordítható egy automata.

7.1. Injektivitás és szürjektivitás

Legyen g egy függvény, ami a következő módon van definiálva: $g: A \longrightarrow B$. Minden $K \subseteq A$ azt mondhatjuk, hogy a K képét kapom, ha a

$$g(K) = \{g(k) | k \in K\},$$

és minden $L \subseteq B$ és az L előképe

$$g^{-1}(L) = \{a \in A | g(a) \in L\}.$$

A $b \in B$

$$g^{-1}(b) = \{a \in A | g(a) = b\}$$

egy halmaz, ami a b előképeit tartalmazza. A g függvényt

- injektív ha B minden elemére igaz hogy legfeljebb egy előképe van:

$$|g^{-1}(b)| \leq 1 \; \forall b \in B$$

- szürjektív ha B minden elemére igaz hogy legalább 1 előképe van:

$$|g^{-1}(b)| \geq 1 \; \forall b \in B$$

- bijektív ha az injektív és a szürjektív tulajdonság is teljesül, vagyis B minden elemére igaz hogy pontosan egy előképe van:

$$|g^{-1}(b)| = 1 \; \forall b \in B$$

A sejtautomata injektív szürjektív vagy bijektív, ha a G leképzési függvény injektív, szürjektív vagy bijektív.

7.2. Megfordítható sejtautomata

Egy sejtautomata G leképzési függvénye megfordítható ha bijektív és a G^{-1} inverz leképzési függvény is sejtautomata leképzési függvény. Vagyis egy A sejtautomata reverzibilis, ha a G leképzési függvény is megfordítható. Az olyan automatát, ami a G^{-1} leképzési függvény segítségével számol, az A automata inverzének fogjuk nevezni és a jelölése A^{-1}. így azt is mondhatjuk, hogy funkcionálisan a két automata egymás inverzei. Az inverz automatára igaz hogy annak az automatának a lépéseit, aminek az inverze visszafelé ismétli el.

Vegyünk egy példát:
Legyen egy egydimenziós automatánk ($d = 1$) ami az 1, 2, 3 állapotokat veheti fel ($S = \{1, 2, 3\}$) és a szomszédsági vektora 2 hosszúságú ($N = (0, 1)$). Az $f(a, b)$ függvény a következő táblázat alapján veszi fel az értékeit.

a \ b	1	2	3
1	1	1	2
2	2	2	1
3	3	3	3

Tétel: *Egy sejtautomata akkor és csak akkor reverzibilis ha injektív.*

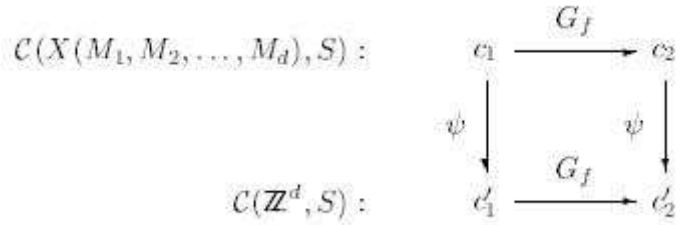
Természetesen egy reverzibilis celluláris automata akkor védi meg az információit, ha a cellákat tartalmazó tömb véges, vagyis ha $X = X(M_1, M_2, \dots, M_d)$. Az automata inverze a leképezési függvény inverzével számol. Ebből következik az a nyilvánvaló tény, hogy a minden sejtautomata ugyanazon a véges tömbökön hajt végre műveleteket pontosan ugyanazon a módon, ahogy egy végtelen konfiguráción hajt végre műveleteket, ami térben periodikus. Ennél egy kicsit pontosabban, a függvény a következő:

$$\psi: C(X(M_1, M_2, \dots, M_d), S) \longrightarrow C(\mathbb{Z}^d, S)$$

Ami a következő módon definiált:

$$\psi(c)(x_1, x_2, \dots, x_d) = c(x_1 \ mod \ M_1, x_2 \ mod \ M_2, \dots, x_d \ mod \ M_d)$$

minden $c \in C(X(M_1, M_2, \dots, M_d), S)$ és $x_1, x_2, \dots, x_d \in \mathbb{Z}$ injektív és a következő diagramnak megfelően fog működni:



7.3. Osztott sejtautomaták

Az osztott sejtautomaták csoportjába tartoznak a reverzibilis sejtautomaták bizonyos típusai. Egy m szomszédos osztott sejtautomata állapothalmaza az $S_1, S_2, \dots, S_m$ véges halmazok Descartes-szorzata.

$$S = S_1 \times S_2 \times \dots \times S_m$$

Az $s_i \in S_i$ elemeket az s állapot komponenseinek fogjuk nevezni. minden cella állapotának i-edik komponense a konfiguráció i-edik sávján található meg. Az automata szabályai az állapot halmaz permutációja lesz.

$$\pi: S \longrightarrow S$$

Minden cella az i-edik komponens i-edik szomszédját tartalmazza, ezeket összeolvasztva az S egy elemévé. ezt egy kicsit pontosabban megfogalmazva: jelöljük s_i^j az j-edik szomszédság i-edik komponensét, és a szabály a következő lesz:

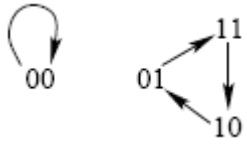
$$f[(s_1^1, s_2^1, \dots, s_m^1), (s_1^2, s_2^2, \dots, s_m^2), \dots, (s_1^m, s_2^m, \dots, s_m^m)] = \pi(s_1^1, s_2^2, \dots, s_m^m).$$

Vegyünk egy példát:

Legyen $d = 1$, $N = (0, 1)$, tehát $m = 2$. Legyen az állapotok halmaza

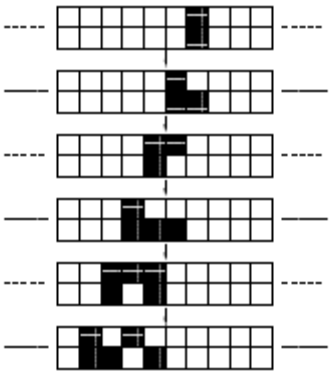
$$S = \{0, 1\} \times \{0, 1\} = \{00, 01, 10, 11\}$$

Az állapot halmazának 4 eleműsége miatt $4! = 24$ különböző permutációja létezik az állapotok halmazának. A 12. ábrán példa automata permutációs térképét láthatjuk.



12. ábra Permutációs térkép

Legyen a kezdő konfiguráció az, hogy egy cella legyen 11 állapotban, az összes többi legyen 00 állapotban. Az osztott sejtautomata egy lépésében a sejtautomaták két egyszerű műveletét hajtjuk végre: először a második sáv egy pozícióval balra csúsztatunk és aztán a π permutációt alkalmazzuk minden cellára. Mindkét művelet injektív vagyis az osztott sejtautomata reverzibilis. Az inverz leképezés először alkalmazza a π^{-1} permutációt minden cellára és aztán a második sávot eggyel jobbra tolja. A 13. ábrán a példának hozott automata grafikus megjelenítése látható. A 0 bit értéket fehér az 1-es bit értéket pedig fekete színnel jelölöm.



13. ábra Osztott sejtautomata grafikusan ábrázolva

Az előzőekből a következő állításokat szűrhetjük le:

Tétel: *az osztott celluláris automaták reverzibilisek.*

7.4. Reverzibilis automaták használata kriptográfiai rendszerként

A reverzibilis sejtautomaták használhatóak kriptográfia rendszerként is. Legyen az $\mathcal{A} = (d, S, N_1, f_1)$ egy reverzibilis celluláris automata, ennek inverze $\mathcal{A}' = (d, S, N_2, f_2)$. A rejtjelezendő szöveget az S szimbólumai segítségével egy d-dimenziós tömbbe fogjuk írni a titkosítás során. Ha a tömb mérete $M_1 \times M_2 \times \ldots \times M_d$, akkor a titkosítandó szöveg egy konfigurációt határoz meg, ami $p \in C(X(M_1, M_2, \ldots, M_d), S)$. A kódolásnak akkor van vége, ha az $\mathcal{A}$ reverzibilis automata p-ben elfogadó állapotba kerül k lépés után. A k szám lehet egy meghatározott pozitív egész, vagy a függhet a tömb méretétől. Az eredményül a c konfigurációban $c = G_{f_1}^k(p)$ a titkosított szöveget kapjuk. A p titkosítatlan szöveg egyszerűen visszanyerhető c-ből az $\mathcal{A}^{-1}$ inverz reverzibilis automata elfogadó állapotával k lépés alatt, vagyis $p = G_{f_2}^k(c)$.

Nyilvánvaló előnye a reverzibilis sejtautomaták kriptográfiai rendszerként való alkalmazásának hogy a lokalitás és a párhuzamosság miatt nagy sebességű hardveres implementációja lehetséges.

7.5.Egy titkos kulcsú kriptográfiai rendszer

Legyen a d korlátlan dimenziók száma és a szomszédsági vektor is legyen hasonló.

$$N = (\overrightarrow{x_1}, \overrightarrow{x_2}, \ldots, \overrightarrow{x_n}).$$

Az állapotok halmaza

$$S = S_1 \times S_2 \times \ldots \times S_n,$$

n véges halmaz Descartes-szorzata, ahol n a szomszédság mérete. Legyen

$$\varphi: S \longrightarrow S$$

korlátlan permutációja az állapotok halmazának. Ezek a jellemzők meghatároznak egy d-dimenziós sejtautomatát. Az $\mathcal{A}_\varphi = (d, S, N, f_\varphi)$ automata szabályait később írjuk fel.

Legyen $\pi_i: S_1 \times S_2 \times \ldots \times S_n \longrightarrow S_i$ ami az S i-edik projekciója lesz. Ez alapján a szabály a következő lesz:

$$f(s_1, s_2, \dots, s_n) = \varphi(\pi_1(s_1), \pi_2(s_2), \dots, \pi_n(s_n))$$

Másképp megfogalmazva, minden cella kap egy S_1 összetevőt az első szomszédságból, egy S_2 összetevőt a második szomszédságból, és így tovább. Az elemekre a φ permutációt alkalmazva egy formulát kapunk ami, az S egy eleme lesz. A formula különböző összetevői n rétegű konfigurációt biztosítanak. Az automata először eltolja ezeket a rétegeket, ezzel létrehozza, a szomszédsági vektort aztán alkalmazza a φ permutációt.

Könnyen beláthatjuk, hogy az $\mathcal{A}$ sejtautomata injektív és egyben reverzibilis is, mi több az $\mathcal{A}^{-1}$ inverz reverzibilis sejtautomata ugyanolyan egyszerű, mint az $\mathcal{A}$. Ennek az automatának a szomszédsági vektora

$$-N = (-\overrightarrow{x_1}, -\overrightarrow{x_2}, \dots, -\overrightarrow{x_n}),$$

az N-ből származik annyi változással hogy minden elem minden koordinátája előjelet vált. A szabályai pedig:

$$g_\varphi(s_1, s_2, \dots, s_n) = \left(\pi_1\left(\varphi^{-1}(s_1)\right), \pi_2\left(\varphi^{-1}(s_2)\right), \dots, \pi_n\left(\varphi^{-1}(s_n)\right)\right).$$

Vagyis az $\mathcal{A}^{-1}$ inverz automata először elfogadja az φ^{-1} inverz permutációt és elrendezi őket az eredeti pozíciójukba.

A fent említett kriptográfiai rendszer abban az értelemben rugalmas, hogy az alkalmazástól függően szabadon megválasztható a d dimenzió, az S állapothalmaz, és az N szomszédsági vektor. Például, ha egy képet akarunk titkosítani, akkor a legtermészetesebb módon a dimenziók számát kettőnek választjuk, mivel a kép téglalap alakú, és minden pixel megfeleltethető egy cellának. Ha 2^m különböző értéket vehet fel egy pixelünk, akkor az állapothalmaz m darab kételemű halmaz Descartes-szorzata, a szomszédsági vektor pedig egy m elemű vektor lesz. Miután meghatároztuk a dimenzió számot, az állapothalmazt, és a szomszédsági vektort, csak a φ permutációs szabályt kell megválasztanunk, ez lesz a titkos kulcs. Meg kell azt jegyeznünk, hogy az s állapotok permutációját listában fogjuk tárolni, és minden ilyen elem $s * \log_2 s$ bit helyet fog elfoglalni.

8. Összegzés

Végignézve a dolgozaton megtudhattuk honnan ered ez az érdekes automata, kik voltak az úttörői milyen állításokat és tételeket tettek a tudományok tudástárába. Láthattuk, hogy a sejtautomata elég hatékony abból a szempontból, hogy ami algoritmikusan leírható, az leírható sejtautomatával is. Néhány példát be is mutattunk a korábbi fejezetekben a teljesség igénye nélkül, de ezen elméleti konstrukció nem ismer határokat, és a párhuzamosságra való igény növekedésével egyre inkább előtérbe kerülhet az alkalmazása a tudomány számos általunk nem is gondolt területén is a jövőben.

Bibliográfia

[1] Martin Gardner: "Mathematical Games: The fantastic combinations of John Conway's new solitaire game "Life"" Scientific American **223**: 120–123 (1970. október),

[2] Konrad Zuse: Calculating Space (1969)

[3] Stephen Wolfram: Statistical Mechanics of Cellular Automata, Rev. Mod. Phys. 55 (1983) 601–644.

[4] Stephen Wolfram: A new kind of science (2002)

[5] Rudy Rucker: The Lifebox, The Seashell and The Soul Thunder’s Mouth Press (2005, október)

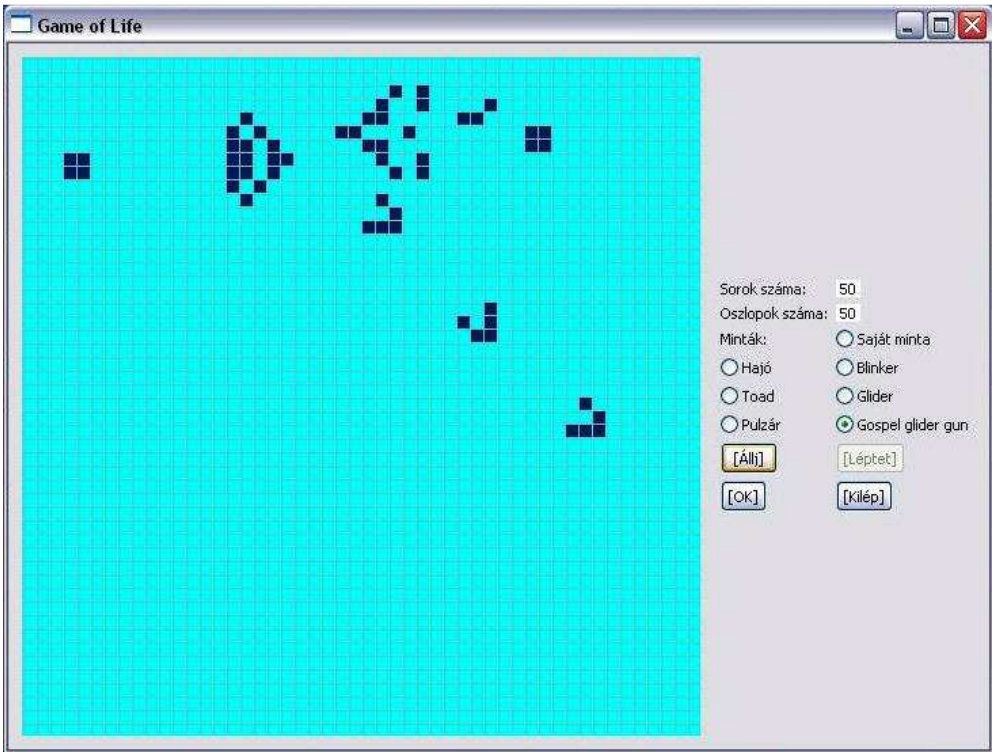
[6] Jarkko Kari: Cellular automata (Lecture notes) University of Tukru (2007)

[7] Jarkko Kari: Cryptosystems based on reversible cellular automata (preprint) University of Turku (1992)

[8] Horváth Géza, Mecsei Zoltán, Nagy Benedek: Gyakorlati összefoglaló Debreceni Egyetem

Függelék

Itt adom közre a dolgozathoz készült programot. A programról tudni kell, hogy folyamatosan és egyesével is lehet léptetni kirajzolt, vagy általunk definiált mintát. A program futtatása Eclipse fejlesztő környezetben ajánlott, és szükséges hozzá az SWT.jar fájl és 4 darab kép is. A képek méretétől nagyban fog függni az ablak mérete. Ajánlott kisméretű, lehetőleg 10x10 pixel nagyságú képet használni, és a képek abszolút elérési útját helyesen megadni. Ezek nélkül a következő osztályok használhatatlanok.



14. ábra A program működés közben

A 14. ábra mutatja a programot működés közben. Ha helyesen adjuk meg a fent felsoroltakat, akkor esetleg színben eltérő, de alapjában véve nagyon hasonló ablakot kell kapnunk futtatáskor.

A program forráskódja a következő:

Automata.java:

```
package automata;
/** Az automatákat megvalósító ősosztály.
*/
public abstract class Automata {
	/**Az automaták szabályait megvalósító metódus őse.
	*/
	public abstract Automata szabály();
}
```

GameOfLife.java:

```
package automata;
/** A Game of Life sejtautomatát megvalósító osztály
*/
public class GameOfLife extends Automata {
	/**A kétdimenziós teret szimuláló adattag. Az adattag véges teret
szimulál.*/
	private int[][] élettér;
	/**Az evolúciót számot tároló adattag. */
	private int evolúció;
	/**Az élettér középső celláját számon tartó adattagok.*/
	private int sk, ok;
	/**Egy üres életteret létrehozó konstruktor. A konstruktor a tér
sorok és az oszlopok számát várja*/
	public GameOfLife(int sor, int oszlop) {
		élettér = new int[sor][oszlop];
		for (int i = 0; i < sor; i++) {
			for (int j = 0; j < oszlop; j++) {
				élettér[i][j] = 0;
			}
		}
		evolúció = 0;
	}
	/**Egy valamilyen definiált mintával rendelkező teret létrehozó
konstruktor. A konstruktor várja a sorok és az oszlopok számát illetve a
minta nevét.*/
	public GameOfLife(int sor, int oszlop, String minta) {
		élettér = new int[sor][oszlop];
		for (int i = 0; i < sor; i++) {
			for (int j = 0; j < oszlop; j++) {
				élettér[i][j] = 0;
			}
		}
		evolúció = 0;
		sk = Math.round(sor / 2);
		ok = Math.round(oszlop / 2);
		if (minta.equals("Hajó"))
			rajzolHajót();
		if (minta.equals("Blinker"))
			rajzolBlinkert();
		if (minta.equals("Toad"))
			rajzolToad();
		if (minta.equals("Glider"))
			rajzolGlider(sor, oszlop);
		if (minta.equals("Pulzár"))
			rajzolPulzar();
		if (minta.equals("Gospel glider gun"))
```

```
                rajzolGliderGun();
      }
      /**Elhelyezi a Hajó nevű mintát a téren.*/
      private void rajzolHajót() {
            this.élettér[sk - 1][ok - 1] = this.élettér[sk - 1][ok] =
this.élettér[sk][ok - 1] = this.élettér[sk][ok + 1] = this.élettér[sk +
1][ok] = 1;
      }
      /**Elhelyezi a Blinker nevű mintát a téren.*/
      private void rajzolBlinkert() {
            this.élettér[sk][ok - 1] = this.élettér[sk][ok] =
this.élettér[sk][ok + 1] = 1;
      }
      /**Elhelyezi a Blinker nevű mintát a téren.*/
      private void rajzolToad() {
            this.élettér[sk][ok] = this.élettér[sk][ok + 1] =
this.élettér[sk][ok + 2] = 1;
            this.élettér[sk + 1][ok - 1] = this.élettér[sk + 1][ok] =
this.élettér[sk + 1][ok + 1] = 1;
      }

      /**Elhelyezi a Glider nevű mintát a téren. A minta tér négy sarkából
indul a tábla kozepe felé.*/
      private void rajzolGlider(int sor, int oszlop) {
            this.élettér[0][1] = 1;
            this.élettér[1][2] = this.élettér[2][0] = this.élettér[2][1] =
this.élettér[2][2] = 1;
            this.élettér[0][oszlop - 2] = 1;
            this.élettér[1][oszlop - 3] = this.élettér[2][oszlop - 3] =
this.élettér[2][oszlop - 2] = this.élettér[2][oszlop - 1] = 1;
            this.élettér[sor - 1][1] = 1;
            this.élettér[sor - 2][2] = this.élettér[sor - 3][0] =
this.élettér[sor - 3][1] = this.élettér[sor - 3][2] = 1;
            this.élettér[sor - 1][oszlop - 2] = 1;
            this.élettér[sor - 2][oszlop - 3] = this.élettér[sor -
3][oszlop - 3] = this.élettér[sor - 3][oszlop - 2] = this.élettér[sor -
3][oszlop - 1] = 1;

      }

      /**Elhelyezi a Pulzár nevű mintát a téren.*/
      private void rajzolPulzar() {
            this.élettér[sk][ok - 2] = this.élettér[sk][ok - 1] =
this.élettér[sk][ok] = this.élettér[sk][ok + 1] = this.élettér[sk][ok + 2]
= 1;
            this.élettér[sk + 1][ok - 2] = this.élettér[sk + 1][ok + 2] =
1;
      }

      /**Elhelyezi a Gospel glider gun nevű mintát a téren.*/
      private void rajzolGliderGun() {
            this.élettér[7][3] = this.élettér[7][4] = this.élettér[8][3] =
this.élettér[8][4] = 1;
            this.élettér[8][11] = this.élettér[9][11] = this.élettér[9][12]
= 1;
            this.élettér[7][12] = this.élettér[7][13] = this.élettér[8][13]
= 1;
            this.élettér[9][19] = this.élettér[10][19] =
this.élettér[11][19] = 1;
```

```
            this.élettér[9][20] = this.élettér[10][21] = 1;
            this.élettér[7][25] = this.élettér[6][25] = this.élettér[7][26]
= 1;
            this.élettér[5][26] = this.élettér[5][27] = this.élettér[6][27]
= 1;
            this.élettér[17][27] = this.élettér[18][27] =
this.élettér[17][28] = 1;
            this.élettér[17][29] = this.élettér[19][28] = 1;
            this.élettér[12][38] = this.élettér[13][38] =
this.élettér[14][38] = 1;
            this.élettér[12][39] = this.élettér[13][40] = 1;
            this.élettér[6][37] = this.élettér[6][38] = this.élettér[5][37]
= this.élettér[5][38] = 1;

      }
      /**Az automata ősosztály szabály metódusát megvalósító metódus.*/
      @Override
      public Automata szabály() {
            int előszomszéd = 0;
            int sor = this.élettér.length, oszlop = this.élettér[0].length;
            GameOfLife új = new GameOfLife(sor, oszlop);
            for (int i = 0; i < sor; i++) {
                  for (int j = 0; j < oszlop; j++) {
                        if (i > 0 && i < sor - 1 && j > 0 && j < oszlop -
1) {
                              for (int k = -1; k <= 1; k++) {
                                    for (int l = -1; l <= 1; l++) {
                                          if ((k != 0 || l != 0)
                                                      && this.élettér[i +
k][j + l] == 1) {
                                                előszomszéd++;
                                          }
                                    }
                              }
                              új.élettér[i][j] =
szomszédVizsgáló(this.élettér[i][j],
                                          előszomszéd);
                              előszomszéd = 0;
                        }
                        if (i % sor == 0 && j > 0 && j < oszlop - 1) {
                              if (this.élettér[i][j - 1] == 1)
                                    előszomszéd++;
                              if (this.élettér[i][j + 1] == 1)
                                    előszomszéd++;
                              if (this.élettér[i + 1][j - 1] == 1)
                                    előszomszéd++;
                              if (this.élettér[i + 1][j] == 1)
                                    előszomszéd++;
                              if (this.élettér[i + 1][j + 1] == 1)
                                    előszomszéd++;
                              új.élettér[i][j] =
szomszédVizsgáló(this.élettér[i][j],
                                          előszomszéd);
                              előszomszéd = 0;
                        }

                        if (i % sor == sor - 1 && j > 0 && j < oszlop - 1)
{
                              if (this.élettér[i][j - 1] == 1)
```

```
                              élőszomszéd++;
                        if (this.élettér[i][j + 1] == 1)
                              élőszomszéd++;
                        if (this.élettér[i - 1][j - 1] == 1)
                              élőszomszéd++;
                        if (this.élettér[i - 1][j] == 1)
                              élőszomszéd++;
                        if (this.élettér[i - 1][j + 1] == 1)
                              élőszomszéd++;
                        új.élettér[i][j] =
szomszédVizsgáló(this.élettér[i][j],
                                    élőszomszéd);
                        élőszomszéd = 0;
                  }

                  if (j % oszlop == 0 && i > 0 && i < sor - 1) {
                        if (this.élettér[i - 1][j] == 1)
                              élőszomszéd++;
                        if (this.élettér[i - 1][j + 1] == 1)
                              élőszomszéd++;
                        if (this.élettér[i][j + 1] == 1)
                              élőszomszéd++;
                        if (this.élettér[i + 1][j] == 1)
                              élőszomszéd++;
                        if (this.élettér[i + 1][j + 1] == 1)
                              élőszomszéd++;
                        új.élettér[i][j] =
szomszédVizsgáló(this.élettér[i][j],
                                    élőszomszéd);
                        élőszomszéd = 0;
                  }
                  if (j % oszlop == oszlop - 1 && i > 0 && i < sor -
1) {
                        if (this.élettér[i - 1][j - 1] == 1)
                              élőszomszéd++;
                        if (this.élettér[i - 1][j] == 1)
                              élőszomszéd++;
                        if (this.élettér[i][j - 1] == 1)
                              élőszomszéd++;
                        if (this.élettér[i + 1][j - 1] == 1)
                              élőszomszéd++;
                        if (this.élettér[i + 1][j] == 1)
                              élőszomszéd++;
                        új.élettér[i][j] =
szomszédVizsgáló(this.élettér[i][j],
                                    élőszomszéd);
                        élőszomszéd = 0;
                  }
                  if (i == 0 && j == 0) {
                        if (this.élettér[i][j + 1] == 1)
                              élőszomszéd++;
                        if (this.élettér[i + 1][j] == 1)
                              élőszomszéd++;
                        if (this.élettér[i + 1][j + 1] == 1)
                              élőszomszéd++;
                        új.élettér[i][j] =
szomszédVizsgáló(this.élettér[i][j],
                                    élőszomszéd);
                        élőszomszéd = 0;
```

```
                        }
                        if (i == sor - 1 && j == 0) {
                              if (this.élettér[i][j + 1] == 1)
                                    élőszomszéd++;
                              if (this.élettér[i - 1][j] == 1)
                                    élőszomszéd++;
                              if (this.élettér[i - 1][j + 1] == 1)
                                    élőszomszéd++;
                              új.élettér[i][j] =
szomszédVizsgáló(this.élettér[i][j],
                                          élőszomszéd);
                              élőszomszéd = 0;
                        }
                        if (i == 0 && j == oszlop - 1) {
                              if (this.élettér[i][j - 1] == 1)
                                    élőszomszéd++;
                              if (this.élettér[i + 1][j] == 1)
                                    élőszomszéd++;
                              if (this.élettér[i + 1][j - 1] == 1)
                                    élőszomszéd++;
                              új.élettér[i][j] =
szomszédVizsgáló(this.élettér[i][j],
                                          élőszomszéd);
                              élőszomszéd = 0;
                        }
                        if (i == sor - 1 && j == oszlop - 1) {
                              if (this.élettér[i][j - 1] == 1)
                                    élőszomszéd++;
                              if (this.élettér[i - 1][j] == 1)
                                    élőszomszéd++;
                              if (this.élettér[i - 1][j - 1] == 1)
                                    élőszomszéd++;
                              új.élettér[i][j] =
szomszédVizsgáló(this.élettér[i][j],
                                          élőszomszéd);
                              élőszomszéd = 0;
                        }

                  }
            }
            új.evolúció++;
            return új;
      }
      /**A tér (i, j) koordinátájú mező tartalmát állítja be.*/
      public void setBit(int i, int j) {
            this.élettér[i][j] = 1;
      }
      /**A tér (i, j) koordinátájú mező tartalmát adja vissza.*/
      public int getBit(int i, int j) {
            return this.élettér[i][j];
      }
      /** A szabály metódus segédmetódusa. Az élő szomszédok segítségével
mondja meg a cellák új állapotát.*/
      private int szomszédVizsgáló(int elem, int élőszomszéd) {
            int vissza = 0;
            if (elem == 1)
                  vissza = (élőszomszéd == 2 || élőszomszéd == 3) ? 1 : 0;
            if (elem == 0)
                  vissza = (élőszomszéd == 3) ? 1 : 0;
```

```
            return vissza;
      }
      /**A Game of Life osztály egy objektumának szöveges
reprezentációja.*/
      public String toString() {
            String s = "";
            for (int i = 0; i < this.élettér.length; i++) {
                  for (int j = 0; j < this.élettér[0].length; j++) {
                        s += this.élettér[i][j];
                  }
                  s += "\n";
            }
            s += "Lépés: " + this.evolúció + "\n";
            return s;
      }
}
```

ControlPanel.java:

```
import org.eclipse.swt.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;
/**A grafikus ablak elemeit definiáló osztály.*/
public class ControlPanel extends Composite {
      /**A tér sorainak számát tároló adattag*/
      int m;
      /**A tér oszlopainak számát tároló adattag.*/
      int n;
      /**A GUI labeljei*/
      Label label, minta, evolucio;
      /**A sorok és az oszlopok számát változtatására használható
szövegmezők.*/
      public Text sor, oszlop;
      /**A léptető indító kilépő és új ablakot letrehozó gombok.*/
      Button clear, start, exit, lepteto;
      /**A minták váltására szolgáló rádió gombokat tároló gombtömb.*/
      Button minták[];
      /**A figyelmeztetések megjelenítésére szolgáló adattag.*/
      MessageBox figyelmeztetés;
      /**A minták neveit tartalmazó String tömb.*/
      String[] mintasor = new String[] { "Saját minta", "Hajó", "Blinker",
                  "Toad", "Glider", "Pulzár", "Gospel glider gun" };
      /**A grafikus interfész elemiet példányosító konstruktor.*/
      public ControlPanel(Composite parent, Display display, int i, int j)
{
            super(parent, SWT.NONE);
            GridLayout layout = new GridLayout(2, false);
            setLayout(layout);
            label = new Label(this, SWT.NONE);
            label.setText("Sorok száma:");
            sor = new Text(this, SWT.SINGLE | SWT.LEFT);
            sor.setText(Integer.toString(i));
            label = new Label(this, SWT.NONE);
            label.setText("Oszlopok száma:");
            oszlop = new Text(this, SWT.SINGLE | SWT.LEFT);
            oszlop.setText(Integer.toString(j));
            layout = new GridLayout(1, false);
            minta = new Label(this, SWT.NONE);
            minta.setText("Minták: ");
```

```
            minták = new Button[mintasor.length];
            for (int k = 0; k < mintasor.length; k++) {
                  minták[k] = new Button(this, SWT.RADIO);
                  minták[k].setText(mintasor[k]);
                  minták[k].pack();
            }
            start = new Button(this, SWT.NONE);
            start.setText("[Indít]");
            lepteto = new Button(this, SWT.NONE);
            lepteto.setText("[Léptet]");
            clear = new Button(this, SWT.NONE);
            clear.setText("[OK]");
            exit = new Button(this, SWT.NONE);
            exit.setText("[Kilép]");
      }
      /**Visszaadja a tér oszlopainak számát*/
      public int getM() {
            return new Integer(sor.getText());
      }
      /**Visszaadja a tér sorainak számát*/
      public int getN() {
            return new Integer(oszlop.getText());
      }
      /**Visszaadja a tér oszlopainak számát*/
      public Button getClear() {
            return clear;
      }
      /**Az indít gombot visszaadó metódus.*/
      public Button getStart() {
            return start;
      }
      /**Az indít gombot visszaadó metódus.*/
      public Label getEvolucio() {
            return evolucio;
      }
      /**A kilépés gombot visszaadó metódus.*/
      public Button getExit() {
            return exit;
      }
      /**A minták kiválasztását segítő rádiógombokat visszaadó metódus.*/
      public Button getMinták(int i) {
            return minták[i];
      }
      /**A figyelmesztetésre szolgáló adattagot visszaadó metódus.*/
      public MessageBox getFigyelmeztetés() {
            return figyelmeztetés;
      }
      /**Az léptető gombot visszaadó metódus.*/
      public Button getLepteto() {
            return lepteto;
      }
}
```

Main.java:

```
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.*;
/**A GUI ablakát létrehozó osztály.*/
```

```
public class Main {

	static int m = 50;
	static int n = 50;
	static int mtemp, ntemp;
	static Tabla tabla;
	static Shell shell;
	static Display display;
	static ControlPanel cp;
	static String[] strings;
	static boolean volt_klikk = false;
	static boolean sync = false;
	static Thread timer;
	static myRunnable timerrunnable;
	static boolean running = false;

	public static void setsync() {
		sync = true;
	}

	public static void resetsync() {
		sync = false;
	}

	public static void main(String[] args) {
		strings = args.clone();
		display = new Display();
		shell = new Shell(display);
		GridLayout layout = new GridLayout(2, false);
		shell.setLayout(layout);
		shell.setText("Game of Life");
		tabla = new Tabla(shell, display, m, n);
		tabla.pack();
		cp = new ControlPanel(shell, display, m, n);
		cp.pack();
		cp.getStart().addListener(SWT.CR, indit);
		cp.getLepteto().addListener(SWT.CR, leptet);
		cp.getClear().addListener(SWT.CR, valtoztat);
		cp.getExit().addListener(SWT.CR, exit);
		cp.getMinták(0).addListener(SWT.MouseUp, minta0);
		cp.getMinták(1).addListener(SWT.MouseUp, minta1);
		cp.getMinták(2).addListener(SWT.MouseUp, minta2);
		cp.getMinták(3).addListener(SWT.MouseUp, minta3);
		cp.getMinták(4).addListener(SWT.MouseUp, minta4);
		cp.getMinták(5).addListener(SWT.MouseUp, minta5);
		cp.getMinták(6).addListener(SWT.MouseUp, minta6);
		shell.pack();
		shell.open();
		while (!shell.isDisposed()) {
			if (running) {
				if (sync) {
					tabla.megvaltoztat();
					display.update();
					resetsync();
				} else if (!display.readAndDispatch())
					display.update();
			} else {
				if (!display.readAndDispatch())
					display.sleep();
```

```
			}
		}
	}
	static Listener exit = new Listener() {
		@Override
		public void handleEvent(Event e) {
			display.dispose();
		}
	};
	static Listener minta0 = new Listener() {
		@Override
		public void handleEvent(Event e) {
			if (m < 5 || n < 5) {
				figyelmeztetes();
				cp.figyelmeztetés.open();
			} else
				tabla = tabla.setTabla("");
		}
	};
	static Listener minta1 = new Listener() {
		@Override
		public void handleEvent(Event e) {
			if (m < 3 || n < 3) {
				figyelmeztetes();
				cp.figyelmeztetés.open();
			} else
				tabla = tabla.setTabla(cp.getMinták(1).getText());
		}
	};
	static Listener minta2 = new Listener() {
		@Override
		public void handleEvent(Event e) {
			if (m < 3 || n < 3) {
				figyelmeztetes();
				cp.figyelmeztetés.open();
			} else
				tabla = tabla.setTabla(cp.getMinták(2).getText());
		}
	};
	static Listener minta3 = new Listener() {
		@Override
		public void handleEvent(Event e) {
			if (m < 5 || n < 5) {
				figyelmeztetes();
				cp.figyelmeztetés.open();
			} else
				tabla = tabla.setTabla(cp.getMinták(3).getText());
		}
	};
	static Listener minta4 = new Listener() {
		@Override
		public void handleEvent(Event e) {
			if (m < 10 || n < 10) {
				figyelmeztetes();
				cp.figyelmeztetés.open();
			} else
				tabla = tabla.setTabla(cp.getMinták(4).getText());
		}
	};
```

```
static Listener minta5 = new Listener() {
	@Override
	public void handleEvent(Event e) {
		if (m < 15 || n < 15) {
			figyelmeztetes();
			cp.figyelmeztetés.open();
		} else
			tabla = tabla.setTabla(cp.getMinták(5).getText());
	}
};
static Listener minta6 = new Listener() {
	@Override
	public void handleEvent(Event e) {
		if (m < 45 || n < 45) {
			figyelmeztetes();
			cp.figyelmeztetés.open();
		} else
			tabla = tabla.setTabla(cp.getMinták(6).getText());
	}
};
static Listener valtoztat = new Listener() {
	public void handleEvent(Event e) {
		if ((e.type == SWT.MouseUp && e.button == 1)
				|| e.type != SWT.MouseUp) {
			mtemp = cp.getM();
			ntemp = cp.getN();
			if (mtemp < 3 || ntemp < 3) {
				figyelmeztetes();
				cp.figyelmeztetés.open();
				cp.oszlop.setText(Integer.toString(n));
				cp.sor.setText(Integer.toString(m));
			} else {
				n = ntemp;
				m = mtemp;
				display.dispose();
				main(strings);
			}
		}
	}
};
static Listener indit = new Listener() {
	public void handleEvent(Event e) {
		if (running) {
			running = false;
			cp.start.setText("[Indít]");
			cp.lepteto.setEnabled(true);
			timerrunnable.stop = true;
			resetsync();
		} else {
			running = true;
			cp.start.setText("[Állj]");
			cp.lepteto.setEnabled(false);
			timerrunnable = new myRunnable();
			timer = new Thread(timerrunnable);
			timer.start();
		}
	}
};
static Listener leptet = new Listener() {
```

```
            @Override
            public void handleEvent(Event e) {
                  tabla.megvaltoztat();
            }
      };

      private static void figyelmeztetes() {
            cp.figyelmeztetés = new MessageBox(shell, SWT.ICON_WARNING);
            cp.figyelmeztetés
                        .setMessage("A tábla túl kicsi a minta
kirajzolásához. Adjon meg más értéket.");
            cp.figyelmeztetés.setText("Game of Life-Figyelmeztetés");
      }
}
```

MyEvent.java:

```
/**A saját event osztály*/
public class MyEvent {
}
```

MyRunnable.java:

```
/**Egy saját szálat megvalósító osztály.*/
public class myRunnable implements Runnable{
        public boolean stop;  public void run() {
            while(!stop){
                    Main.setsync();
                        try{
                        Thread.sleep(100);
                        }
                        catch(InterruptedException ex){
                                System.out.println(ex.toString());
                        }
            }
        }

}
```

Tabla.java:

```
import org.eclipse.swt.*;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;
import automata.*;

/**A táblát megvalósító osztály.*/
public class Tabla extends Composite {
      /**A tábla oszlopainak számát tároló adattag.*/
      int m;
      /**A tábla sorainak számát tároló adattag.*/
      int n;
      /**A kirajzolást segítő kép.*/
      static Image image;
      /**A kirajzolást segítő kép.*/
      static Image image2;
      /**A kirajzolást segítő kép.*/
      static Image image3;
```

```
	/**A kirajzolást segítő kép.*/
	static Image image4;
	/**A Game of Life objektumot tároló adattag.*/
	private static GameOfLife gol;
	/**A táblát példányosító konstruktor. A képekhez értékeket rendel,
példányosít egy Game of Life objektumot, és a Game of Life objektumnak
megfelelően létrehozza a táblát. Illetve a konstruktor listenereket
tartalmaz, ami segítségével saját mintát is definiálhatunk.*/
	public Tabla(Composite parent, Display display, int n, int m) {
		super(parent, SWT.NONE);
		this.m = m;
		this.n = n;
		gol = new GameOfLife(m, n);
		GridLayout layout = new GridLayout(m, false);
		layout.horizontalSpacing = 0;
		layout.verticalSpacing = 0;
		setLayout(layout);
		image = new Image(display,

	"D:\\gol\\workspace\\Tablajatek\\lib\\img\\image1.bmp");
		image2 = new Image(display,

	"D:\\gol\\workspace\\Tablajatek\\lib\\img\\image2.bmp");
		image3 = new Image(display,

	"D:\\gol\\workspace\\Tablajatek\\lib\\img\\image3.bmp");
		image4 = new Image(display,

	"D:\\gol\\workspace\\Tablajatek\\lib\\img\\image4.bmp");
		for (int i = 0; i < n * m; i++) {
			final Label label = new Label(this, SWT.NONE);
			final int j = Math.round(i / m), k = i % n;
			if (gol.getBit(j, k) == 1) {
				label.setImage(image3);
			} else {
				label.setImage(image);
			}
			label.addListener(SWT.MouseEnter, new Listener() {
				public void handleEvent(Event e) {
					if (label.getImage().equals(image))
						label.setImage(image2);
					else if (label.getImage().equals(image3))
						label.setImage(image4);
				}
			});
			label.addListener(SWT.MouseExit, new Listener() {
				public void handleEvent(Event e) {
					if ((label.getImage().equals(image2)))
						label.setImage(image);
					else if ((label.getImage().equals(image4)))
						label.setImage(image3);
				}
			});
			label.addListener(SWT.MouseDown, new Listener() {
				public void handleEvent(Event e) {
					if (label.getImage().equals(image2)
							|| label.getImage().equals(image))
{
						label.setImage(image3);
```

```
                                    gol.setBit(j, k);
                              }
                        }
                  });
            }
            ;
      }
      /**Egy tábla objektummal visszatérő beállító metódus. Példányosít egy
Game of Life objektumot és kirajzolja azt.*/
      public Tabla setTabla(String s) {
            gol = new GameOfLife(n, m, s);
            kirajzol(gol);
            return this;
      }
      /**A Game of Life objektum kirajzolását elvégző metódus.*/
      public void kirajzol(GameOfLife gol) {
            int j, k;
            Control elemek[];
            elemek = this.getChildren();
            for (int i = 0; i < n * m; i++) {
                  j = Math.round(i / m);
                  k = i % m;
                  if (gol.getBit(j, k) == 1) {
                        ((Label) elemek[i]).setImage(image3);

                  } else {
                        ((Label) elemek[i]).setImage(image);
                  }
            }
      }
      /**A Game of Life objektum megváltoztatását és kirajzolását elvégző
metódus. A Game of Life objektumra meghívja a szabály metódust, majd azt
kirajzolja.*/
      public void megvaltoztat() {
            gol = (GameOfLife) gol.szabály();
            kirajzol(gol);
      }
      /**A tábla méreteit és a Game of Life objektum szöveges
megvalósítása.*/
      public String toString() {
            String s = "";
            s += "a sorok száma " + m + "\n";
            s += "az oszlopok száma " + n + "\n";
            s += gol.toString();
            return s;
      }
}
```